**XML, DTD, and XPath**

CPS 116
Introduction to Database Systems

---

**Announcements**

- Midterm has been graded
  - Graded exams available in my office
  - Grades posted on Blackboard
  - Sample solution and score distribution emailed

---

**From HTML to XML (eXtensible Markup Language)**

- HTML describes the presentation of the content
  ```xml
  <h1>Bibliography</h1>
  <p><i>Foundations of Databases</i>
  Abiteboul, Hull, and Vianu
  Addison Wesley, 1995
  </p>...
  ```
- XML describes only the content
  ```xml
  <bibliography>
  <book title="Foundations of Databases" author="Abiteboul" author="Hull" author="Vianu"
  publisher="Addison Wesley" year="1995">
  
  </book>...
  </bibliography>
  ```
- Separation of content from presentation simplifies content extraction and allows the same content to be presented easily in different looks

---

**Other nice features of XML**

- Portability: Just like HTML, you can ship XML data across platforms
  - Relational data requires heavy-weight protocols, e.g., JDBC
- Flexibility: You can represent any information (structured, semi-structured, documents, …)
  - Relational data is best suited for structured data
- Extensibility: Since data describes itself, you can change the schema easily
  - Relational schema is rigid and difficult to change

---

**XML terminology**

- Tag names: book, title, ...
- Start tags: `<book>`, `<title>`, ...
- End tags: `</book>`, `</title>`, ...
- An element is enclosed by a pair of start and end tags: `<book>...</book>
- Elements can be nested: `<book>...</title>...</book>`
- Empty elements: `<is_textbook/>`
- Can be abbreviated: `<i>`
- Elements can also have attributes: `<book ISBN="" price="80.00">`

---

**Well-formed XML documents**

A well-formed XML document
- Follows XML lexical conventions
  - Wrong: `<section>We show that x < 0.</section>`
  - Right: `<section>We show that x &lt; 0.</section>`
    - Other special entities: `&gt;` becomes `&` and `&` becomes `&amp;`
- Contains a single root element
- Has tags that are properly matched and elements that are properly nested
  - Right: `<section>...</subsection>...</section>`
  - Wrong: `<section>...</subsection>...</subsection>...</section>`

---
More XML features

- Comments: <!-- Comments here -->
- CDATA: <![CDATA[Tags: <book>,...]]>
- ID’s and references
  
  ```xml
  <person id="o12"><name>Homer</name>…</person>
  <person id="o34"><name>Marge</name>…</person>
  <person id="o56" father="o12" mother="o34"><name>Bart</name>…</person>…
  ```
- Namespaces allow external schemas and qualified names
  
  ```xml
  <book xmlns:myCitationStyle="http://…/mySchema">
    <myCitationStyle:title>…</myCitationStyle:title>
    <myCitationStyle:author>…</myCitationStyle:author>…
  </book>
  ```
- Processing instructions for apps: <? ... java applet ... ?>
- And more...

Valid XML documents

- A valid XML document conforms to a Document Type Definition (DTD)
  - A DTD is optional
  - A DTD specifies
    - A grammar for the document
    - Constraints on structures and values of elements, attributes, etc.
- Example
  
  ```xml
  <!DOCTYPE bibliography [ 
    <!ELEMENT bibliography (book+)>
    <!ELEMENT book (title, author*, publisher?, year?, section*)>
    <!ATTLIST book ISBN ID #REQUIRED>
    <!ATTLIST book price CDATA #IMPLIED>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT author (#PCDATA)>
    <!ELEMENT publisher (#PCDATA)>
    <!ELEMENT year (#PCDATA)>
    <!ELEMENT section (title, (#PCDATA)?, section*)>
  ]>
  ```

DTD explained

- **<!DOCTYPE bibliography [**
  - bibliography is the root element of the document
  - bibliography contains a sequence of one or more book elements
  - **<!ELEMENT book (title, author*, publisher?, year?, section*)>**
    - book consists of a title, zero or more authors, an optional publisher, and zero or more sections, in sequence
  - **<!ATTLIST book ISBN ID #REQUIRED>**
    - book has a required ISBN attribute which is a unique identifier
  - **<!ATTLIST book price CDATA #IMPLIED>**
    - book has an optional (#IMPLIED) price attribute which contains character data

Other attribute types include IDREF (reference to an ID), IDREFS (space-separated list of references), enumerated list, etc.

DTD explained (cont’d)

- **<!ELEMENT title (#PCDATA)>**
  - PCDATA is text that will be parsed (<...> will be treated as markup tags, and &lt; etc. will be treated as entities)
  - A DTD is a character data that contains parsed character data (#PCDATA)

- **<!ELEMENT author (#PCDATA)>**
- **<!ELEMENT publisher (#PCDATA)>**
- **<!ELEMENT year (#PCDATA)>**
- **<!ELEMENT section (title, (#PCDATA)?, section*)>**
  - Each section starts with a title, followed by some optional text and then zero or more subsections

Using DTD

- **<!DOCTYPE bibliography [**
  - bibliography is the root element of the document
  - bibliography contains a sequence of one or more book elements
  - **<!ELEMENT book (title, author*, publisher?, year?, section*)>**
    - book consists of a title, zero or more authors, an optional publisher, and zero or more sections, in sequence
  - **<!ATTLIST book ISBN ID #REQUIRED>**
    - book has a required ISBN attribute which is a unique identifier
  - **<!ATTLIST book price CDATA #IMPLIED>**
    - book has an optional (#IMPLIED) price attribute which contains character data

Other attribute types include IDREF (reference to an ID), IDREFS (space-separated list of references), enumerated list, etc.

Why use DTD’s?

- Benefits of not using DTD
  - Unstructured data is easy to represent
  - Overhead of DTD validation is avoided
- Benefits of using DTD
  - DTD can serve as a schema for the XML data
    - Guards against errors
    - Helps with processing
  - DTD facilitates information exchange
    - People can agree to use a common DTD to exchange data (e.g., XHTML)
XML versus relational data

Relational data
- Schema is always fixed in advance and difficult to change
- Simple, flat table structures
- Ordering of rows and columns is unimportant
- Data exchange is problematic
- “Native” support in all serious commercial DBMS

XML data
- Well-formed XML does not require predefined, fixed schema
- Nested structure; ID/IDREF(S) permit arbitrary graphs
- Ordering forced by document format; may or may not be important
- Designed for easy exchange
- Often implemented as an “add-on” on top of relations

Query languages for XML

- XPath
  - Path expressions with conditions
    - Building block of other standards (XQuery, XSLT, XPointer, etc.)
- XQuery
  - XPath + full-fledged SQL-like query language
- XSLT
  - XPath + transformation templates

Example DTD and XML

```xml
<?xml version="1.0"?>
<!DOCTYPE bibliography [
<!ELEMENT bibliography (book+)>
<!ELEMENT book (title, author*, publisher?, year?, section*)>
<!ATTLIST book ISBN CDATA #REQUIRED>
<!ATTLIST book price CDATA #IMPLIED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT section (title, (#PCDATA)?, section*)>
]>
<bibliography>
  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
    <section>…</section>…
  </book>
…
</bibliography>
```

A tree representation

XPath

- XPath specifies path expressions that match XML data by navigating down (and occasionally up and across) the tree
- Example
  - Query: /bibliography/book/author
    - Like a UNIX path
  - Result: all author elements reachable from root via the path /bibliography/book/author

Basic XPath constructs

/ separator between steps in a path
name matches any child element with this tag name
* matches any child element
@name matches the attribute with this name
@* matches any attribute
// matches any descendant element or the current element itself
  . matches the current element
  .. matches the parent element
Simple XPath examples

- All book titles
  /bibliography/book/title
- All book ISBN numbers
- All title elements, anywhere in the document
  //title
- All section titles, anywhere in the document
  //section/title
- Authors of bibliographical entries (suppose there are articles, reports, etc. in addition to books)
  /bibliography/*/author

Predicates in path expressions

- [condition] matches the current element if condition evaluates to true on the current element
- Books with price lower than $50
  /bibliography/book[@price<50]
- Books with author “Abiteboul”
  /bibliography/book[@author='Abiteboul']
- Books with a publisher child element
  /bibliography/book/publisher
- Prices of books authored by “Abiteboul”
  /bibliography/book[@author='Abiteboul']/price

More complex predicates

- Predicates can have and's and or's
- Books with price between $40 and $50
  /bibliography/book[40<=$price and $price<=50]
- Books authored by “Abiteboul” or those with price lower than $50
  /bibliography/book[author='Abiteboul' or @price<50]

Predicates involving node-sets

- /bibliography/book[author='Abiteboul']
- There may be multiple authors, so author in general returns a node-set (in XPath terminology)
- The predicate evaluates to true as long as it evaluates true for at least one node in the node-set, i.e., at least one author is “Abiteboul”
- Tricky query
  /bibliography/book[author='Abiteboul' and author!='Abiteboul']
  - Will it return any books?

XPath operators and functions

- Frequently used in conditions:
  x + y, x - y, x * y, x div y, x mod y
- contains(x, y) true if string x contains string y
- count(node-set) counts the number nodes in node-set
- position() returns the position of the current node in the currently selected node-set
- last() returns the size of the currently selected node-set
- name() returns the tag name of the current element

More XPath examples

- All elements whose tag names contain “section” (e.g., “subsection”)
  /*[contains(name(), 'section')]*/
- Title of the first section in each book
  /bibliography/book/section[position()==1]/title
  - A shorthand: /bibliography/book/section[1]/title
- Title of the last section in each book
  /bibliography/book/section[position()==last()]/title
- Books with fewer than 10 sections
  /bibliography/book[count(section)<10]
- All elements whose parent’s tag name is not “book”
  /*[name()]!='book'*/
A tricky example

- Suppose that `price` is a child element of `book`, and there may be multiple prices per book.
- Books with some price in range \([20, 50]\)
  - How about:\n    
    `/bibliography/book`\n    
    `[price >= 20 and price <= 50]`
  - Correct answer:\n    
    `/bibliography/book`\n    
    `[price[. >= 20 and . <= 50]]`

De-referencing IDREF's

- \(\text{id(identifier)}\) returns the element with the unique `identifier`.
- Suppose that books can make references to other books.
  
  `<section><title>Introduction</title>`\n  
  `XML is a hot topic these days; see <bookref ISBN="ISBN-10"/> for more details…`\n  
  `</section>`
- Find all references to books written by "Abiteboul" in the book with "ISBN-10":
  
  `/bookref[id(@ISBN)/author='Abiteboul']`

General XPath location steps

- Technically, each XPath query consists of a series of location steps separated by `/`.
- Each location step consists of:
  - An axis: `self`, `attribute`, `parent`, `child`, `ancestor`, `ancestor-or-self`, `descendent`, `descendent-or-self`, `following`, `following-sibling`, `preceding`, `preceding-sibling`, and `namespace`.
  - A node test: either a name test (e.g., `book`, `section`, `*`) or a type test (e.g., `text()`, `node()`, `comment()`), separated from the axis by `::`.
  - Zero or more predicates (or conditions) enclosed in square brackets.

Example of verbose syntax

Verbose (axis, node test, predicate):

`/child::bibliography`\n
`/descendent-or-self::node()`\n
`/child::title`

Abbreviated:


- `child` is the default axis.
- `/` stands for `/descendent-or-self::node()`.